JAGS Prediction Function Tutorial

Marco Salvo

2025-04-09

Please reach out to me at marco.salvo@umconnect.umt.edu with any issues, bugs, or update ideas

The JAGS Prediction Function

Below is a guide and tutorial for using my homemade JAGS prediction function, which allows the automated prediction of simulation results from JAGS models.

This function was built to be used with output from the jagsUI package, so I'm unsure how it would work with Rjags or other packages, but as long as you have simulation data outputs, it should work.

Let's dig into the function with the whole function being below:

```
# predict from jags output
jags.predict <- function(intercepts,
                         slope.coefficients,
                         exp.cov.values,
                         credible.intervals = c(0.025, 0.5, 0.975),
                         res = 100,
                         link.function,
                         lognormal.sd,
                         covariate.type = "continuous"
){
  #extract the number of iterations from your simulated intercepts
  iters <- length(intercepts)

  ### Setup Data:

  #contin
  if(covariate.type == "continuous"){
    #create sequences of your covariate values
    for(ixi in 1:length(exp.cov.values)){ #for every covariate value
      if("numeric" %in% class(exp.cov.values[[ixi]])){ # if it is numeric:
        if(length(exp.cov.values[[ixi]]) == 1){ #if expected value is length 1, repeat it up the times
          exp.cov.values[[ixi]] <- rep(exp.cov.values[[ixi]], times = res)
        } else if(length(exp.cov.values[[ixi]]) == 2){ #if expected values are length 2 (a range), crea
          exp.cov.values[[ixi]] <- seq(exp.cov.values[[ixi]][1],
                                       exp.cov.values[[ixi]][2],
                                       length.out = res)
        } else { #if a covariate value is not length 1 or 2, give an error
          stop("Expected covariate values must be of length 1 or 2")
        }
```

```
    }
  }
} else if(covariate.type == "categorical"){

  #check if the name of each list item is "category"
  is.category <- sapply(exp.cov.values, function(x) x == "category")

  #for every covariate value
  for(ixi in 1:length(exp.cov.values)){
    if(is.category[ixi] == T){ #if it is categorical
      exp.cov.values[[ixi]] <- rep(0, times = length(exp.cov.values[is.category]) + 1) #create a sequ
    } else if(is.category[ixi] == F){ #else if it's continuous
      if(length(exp.cov.values[[ixi]]) == 1){
        exp.cov.values[[ixi]] <- rep(exp.cov.values[[ixi]],
                                     times = length(exp.cov.values[is.category]) + 1) #repeat the mea
      } else { #no varying values within a categorical prediction (yet)
        stop("Current version does not account for interactions, set continuous covariates to their m
      }
    }
  }

  #Fill in 1's in every 0 sequence
  for(i in seq_along(exp.cov.values)){
    if(i == 1){pos <- 2} #set position index to 2 for the first value

    if (all(exp.cov.values[[i]] == 0)) {  # Check if the vector is all zeros
      if(i != 1){pos <- pos + 1} #add a 1 to the position index
      exp.cov.values[[i]][pos] <- 1 #set the position value to 1
    }
  }

  #set res
  res <- length(exp.cov.values[is.category]) + 1

} else {stop("covariate.type must be one of 'continuous' or 'categorical'.")}


#create a matrix for:
pred.mat <- matrix(NA, iters, res) #predicted values
pred.cred <- matrix(NA, res, length(credible.intervals)) #predicted median and credible intervals

# Get your estimates
for(jlv in 1:res){ # for all resolutions:

  parameter.values <- list() #initialize list to store parameter values in
  for(ixi in 1:length(exp.cov.values)){ #for all covariates in the model:

    if("numeric" %in% class(exp.cov.values[[ixi]])){ #different indexing for vectors and matrices
      parameter.values[[ixi]] <- slope.coefficients[[ixi]] * exp.cov.values[[ixi]][jlv]#get the produ
    } else if("matrix" %in% class(exp.cov.values[[ixi]])){
      parameter.values[[ixi]] <- slope.coefficients[[ixi]] * exp.cov.values[[ixi]][,jlv]#get the prod
    } else {
      stop("expected values must be a numeric vector or a expected.response.matrix matrix derived fro
```

```
      }


      if(ixi == 1){ #for the first value, add the intercepts and the first covariate term
        pred.mat[,jlv] <- intercepts + parameter.values[[ixi]]
      } else { #for all other values, add them to the current matrix column to get the complete expecte
        pred.mat[,jlv] <- pred.mat[,jlv] + parameter.values[[ixi]]
      }

    }

    #If the model is not following a normal distribution:
    if(link.function == "log"){ #for log links
      pred.mat[,jlv] <- exp(pred.mat[,jlv])
    } else if(link.function == "lognormal"){ #for log-normal links
      pred.mat[,jlv] <- rlnorm(iters, pred.mat[,jlv], lognormal.sd)
    } else if(link.function == "logit"){ # for logit link
      pred.mat[,jlv] <- exp(pred.mat[,jlv])/(1+exp(pred.mat[,jlv]))
    }

    #get predicted credible intervals
    pred.cred[jlv,] <- quantile(pred.mat[,jlv], credible.intervals)
  }

  #melted observations for graphing
  melted.obs <- melt(pred.mat); names(melted.obs) <- c('iteration','cov','response')

  return(list(expected.cov.values = exp.cov.values,
              expected.response.matrix = pred.mat,
              expected.quantiles = pred.cred,
              plotting.data = melted.obs))

}
```

Arguements

There's obviously a lot that goes into the function so let's just understand the steps that go into it. First let's start with an overview of the arguments:

intercepts: The simulated model intercepts. Should be a numeric vector

slope.coefficients: The simulated slope coefficients of all slope parameters included in your model. This should be a list of numeric vectors, even if your model only has one slope coefficient.

exp.cov.values: The covariate values to be predicted over. This should be a list of the same length of slope.coefficients with values in the same order with their corresponding slope coefficients. For covariate.type = "continuous", the items in list should either be a single numeric value, which will be repeated the number of times in res, or a numeric vector of length 2 which represent the minimum and maximum values of a covariate which you want to predict over and will be turned into a sequence the length out of res. For covariate.type = "categorical",the list items should either be a character vector named "category" for the categorical slope positions, or a single numeric value either representing the mean of any continuous covariate or 0 for categorical covariates not being predicted over (say, if you have two different categorical groupings in your model, such as land cover and time of day [day vs. night], but only want to predict land cover, the positions with the slope coefficient for night would = 0).

credible.intervals: A numeric vector of credible intervals to be predicted over. Default is median and 95% credible intervals.

res: The desired resolution of your output. The default is 100, which usually produces good graphs

link.function: The link function used in your model as a character. Currently supported link functions include normal ("identity"), log ("log"), logit ("logit"), and lognormal ("lognormal").

lognormal.sd: Only needed for link.function = "lognormal", a numeric vector of simulated standard deviation values from your model.

covariate.type: Is the covariate being predicted "continuous" (default) or "categorical"

Outputs

The output from these models is composed of a list with four items:

expected.cov.values: The full sequence of all expected covariate values fed to the model. Listed in the same order as the list given to *exp.cov.values* argument of the function.

expected.response.matrix: The matrix of expected responses for each iteration for each of the covariate values provided based on your model formula.

expected.quantiles: The expected value of each quantile given in the *quantiles* argument at each expected covariate value.

plotting.data: A melted matrix which contains the values for plotting a smooth scatter plot. "iterations" is the iteration number of the data, "cov" is the specific covariate index location, and "response" is the expected response value for that iteration and covariate index value

The Inner Workings

Now that we understand the arguments and outputs of the function, let's break down what is actually going on inside of this function.

1) The values provided to the *exp.cov.values* are made into a sequence which is the length of the value provided to *res* for continuous covariates. If only one value is provided, it is repeated that many times. If you provide the *expected.response.matrix* output of this function to the *exp.cov.values* for structural equation modeling predictions, it will keep the matrix as is. For categorical covariates, it makes a sequence of 0's which is the length of the number of categories where one value in the sequence equals 1 in each category to make a prediction for each category. The position of the one is different for each category to ensure there is no overlap.

2) For your model, every slope coefficient gets multiplied by each value of it's corresponding sequence of covariate values. Those all get added to the intercept, and then transformed using your chosen link function. This happens for every iteration to create a predicted matrix of response values.

3) We then take the predicted values, and get the quantiles for the response at each covariate value.

4) We then melt the predicted response matrix into a 3 column matrix found in *plotting.data* for easier use in plots

It's really not that hard, but also simplifies this task a ton!

Examples

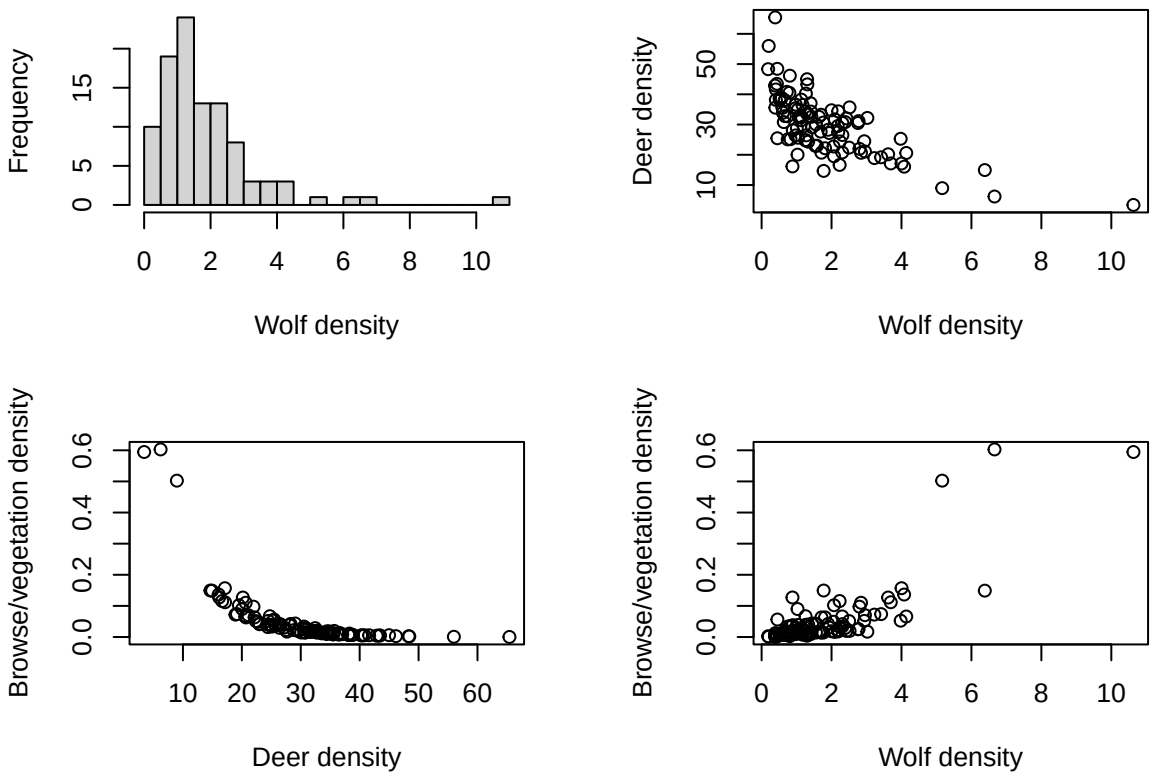
Now lets show how it can be used:

Let's load the packages we'll need for this below:

```
library(jagsUI)
library(reshape2)
```

First let's simulate some simple data predicting the response of deer density to wolf density, and vegetation density to deer density. This simulation and JAGS code is borrowed from a structural equation modeling class taught by Dr. Thomas Riecke:

```
#set seed
set.seed(14763)

# 100 sites
n <- 100

#wolf density
wolf <- rlnorm(n, 0.35, 0.75)

#Deer density
alpha0 = 3.75
alpha1 = -0.225
deer <- rlnorm(n, alpha0 + alpha1 * wolf, 0.25)

#Vegetation density
beta0 <- 0
beta1 <- -0.125
veg <- rlnorm(n, beta0 + beta1 * deer, 0.25)

par(mfrow = c(2,2), mar = c(5.1,5.1,2.1,2.1))
hist(wolf, xlab = 'Wolf density',
     breaks = 25, main = NULL)
plot(deer ~ wolf, ylab = 'Deer density', xlab = 'Wolf density')
plot(veg ~ deer, ylab = 'Browse/vegetation density', xlab = 'Deer density')
plot(veg ~ wolf, ylab = 'Browse/vegetation density', xlab = 'Wolf density')
```



We can see with our simulated data, there is a clear relationship between wolves, deer, and vegetation. So let's model this in a structural equation model in JAGS using a path anlaysis, since wolves likely don't have a direct effect on vegetation, but affect it through the regulation of deer density.

We'll build a models where:

1) deer ~ wolves
2) veg ~ deer

We'll use a lognormal distribution since that's how we simulated our data and vague priors.

```
sink("m_path.jags")
cat("
    model {

    # priors for everything assigned in a loop to minimize unnecessary coding mistakes
    for (j in 1:2){
      alpha[j] ~ dnorm(0, 0.1)
      beta[j] ~ dnorm(0, 0.1)
      sigma[j] ~ dgamma(1,1)
      tau[j] = 1/(sigma[j] * sigma[j])
    }

    for (i in 1:n){
      deer[i] ~ dlnorm(alpha[1] + alpha[2] * wolf[i], tau[1])
      veg[i] ~ dlnorm(beta[1] + beta[2] * deer[i], tau[2])
```

```
    }

    }
    ",fill = TRUE)
sink()
```

Next we'll run our model:

```
jags.data <- list(n = n, wolf = wolf, deer = deer, veg = veg)
inits <- function(){list()}
parameters <- c('alpha','beta','sigma')

# number of chains (nc), thinning rate (nt), number of iterations (ni), and number to burn-in (nb)
nc <- 4
nt <- 10
ni <- 20000
nb <- 10000

model <- jags(jags.data, inits, parameters, "m_path.jags", parallel = T,
              n.chains = nc, n.thin = nt, n.iter = ni, n.burnin = nb)
```

We can see the model results below:

```
print(model)
```

```
## JAGS output for model 'm_path.jags', generated by jagsUI.
## Estimates based on 4 chains of 20000 iterations,
## adaptation = 100 iterations (sufficient),
## burn-in = 10000 iterations and thin rate = 10,
## yielding 4000 total samples from the joint posterior.
## MCMC ran in parallel for 0.084 minutes at time 2024-12-04 11:04:46.745625.
##
##               mean    sd     2.5%      50%    97.5% overlap0     f  Rhat n.eff
## alpha[1]     3.860 0.037    3.790    3.859    3.932    FALSE 1.000 1.000  4000
## alpha[2]    -0.245 0.015   -0.274   -0.245   -0.216    FALSE 1.000 1.001  4000
## beta[1]      0.014 0.070   -0.126    0.014    0.150     TRUE 0.581 1.003   759
## beta[2]     -0.126 0.002   -0.130   -0.126   -0.122    FALSE 1.000 1.002  1039
## sigma[1]     0.246 0.018    0.214    0.245    0.284    FALSE 1.000 1.000  4000
## sigma[2]     0.242 0.018    0.211    0.241    0.280    FALSE 1.000 1.000  4000
## deviance -143.513 3.610 -148.412 -144.259 -134.893    FALSE 1.000 1.000  4000
##
## Successful convergence based on Rhat values (all < 1.1).
## Rhat is the potential scale reduction factor (at convergence, Rhat=1).
## For each parameter, n.eff is a crude measure of effective sample size.
##
## overlap0 checks if 0 falls in the parameter's 95% credible interval.
## f is the proportion of the posterior with the same sign as the mean;
## i.e., our confidence that the parameter is positive or negative.
##
## DIC info: (pD = var(deviance)/2)
## pD = 6.5 and DIC = -136.991
## DIC is an estimate of expected predictive error (lower is better).
```

Now that we've gone through those steps, let's predict some data with our function. We'll start by predicting the effect of wolf density on deer density.

First let's extract the values we need from the model:

```
m.iters <- length(model$sims.list$alpha[,1]) #number of model iterations
m.intercepts <- model$sims.list$alpha[,1] #The intercepts of our model
m.slopes <- list(model$sims.list$alpha[,2]) #slopes of our model, remember, this MUST be in a list form
m.covs <- list(c(min(wolf), max(wolf))) #The range of covariate values to predict across, also MUST be
m.sd <- model$sims.list$sigma[,1] #The simulated standard deviations, needed for the lognormal distribu
```

And now lets run the predictions:

```
wolfdeer.predict <-
jags.predict(intercepts = m.intercepts,
             slope.coefficients = m.slopes,
             exp.cov.values = m.covs,
             credible.intervals = c(0.025,0.5,0.975),
             res = 100,
             link.function = "lognormal",
             lognormal.sd = m.sd)
```

Lets examine the outputs from this.

First we have the plotting data:

```
wolfdeer.predict$plotting.data[1:10,]
```

```
##    iteration cov response
## 1          1   1 41.95938
## 2          2   1 42.25819
## 3          3   1 60.47869
## 4          4   1 44.82128
## 5          5   1 34.72257
## 6          6   1 51.69282
## 7          7   1 48.14808
## 8          8   1 34.17816
## 9          9   1 45.31483
## 10        10   1 50.54176
```

Next we have the expected covariate values, which you can see have been transformed to a numeric vector of 100 values

```
wolfdeer.predict$expected.cov.values
```

```
## [[1]]
##   [1]  0.1874511  0.2929630  0.3984749  0.5039868  0.6094987  0.7150106
##   [7]  0.8205225  0.9260344  1.0315463  1.1370582  1.2425701  1.3480820
##  [13]  1.4535939  1.5591058  1.6646177  1.7701296  1.8756415  1.9811534
##  [19]  2.0866653  2.1921772  2.2976891  2.4032010  2.5087129  2.6142248
##  [25]  2.7197367  2.8252486  2.9307605  3.0362724  3.1417843  3.2472961
##  [31]  3.3528080  3.4583199  3.5638318  3.6693437  3.7748556  3.8803675
##  [37]  3.9858794  4.0913913  4.1969032  4.3024151  4.4079270  4.5134389
```

```
##  [43]  4.6189508  4.7244627  4.8299746  4.9354865  5.0409984  5.1465103
##  [49]  5.2520222  5.3575341  5.4630460  5.5685579  5.6740698  5.7795817
##  [55]  5.8850936  5.9906055  6.0961174  6.2016293  6.3071412  6.4126531
##  [61]  6.5181649  6.6236768  6.7291887  6.8347006  6.9402125  7.0457244
##  [67]  7.1512363  7.2567482  7.3622601  7.4677720  7.5732839  7.6787958
##  [73]  7.7843077  7.8898196  7.9953315  8.1008434  8.2063553  8.3118672
##  [79]  8.4173791  8.5228910  8.6284029  8.7339148  8.8394267  8.9449386
##  [85]  9.0504505  9.1559624  9.2614743  9.3669862  9.4724981  9.5780100
##  [91]  9.6835219  9.7890337  9.8945456 10.0000575 10.1055694 10.2110813
##  [97] 10.3165932 10.4221051 10.5276170 10.6331289
```

Next we have the prediction matrix. I've shortened the output so it doesn't take up this whole pdf, but you can see it has the same dimensions as the number of iterations and the resolution provided to jags.predict.

```
print(dim(wolfdeer.predict$expected.response.matrix))
```

```
## [1] 4000  100
```

```
wolfdeer.predict$expected.response.matrix[1:10, 1:4]
```

```
##           [,1]     [,2]     [,3]     [,4]
##  [1,] 41.95938 32.18277 38.26187 39.76882
##  [2,] 42.25819 74.35310 54.25540 58.89005
##  [3,] 60.47869 40.29105 25.74116 62.01228
##  [4,] 44.82128 32.95588 27.74147 57.70703
##  [5,] 34.72257 59.46872 51.28360 35.00643
##  [6,] 51.69282 25.95007 54.77636 41.79236
##  [7,] 48.14808 39.11437 38.74574 47.70678
##  [8,] 34.17816 65.27136 25.78580 41.04276
##  [9,] 45.31483 33.16439 36.87363 30.27715
## [10,] 50.54176 37.48512 31.35782 35.13155
```

And finally we have our quantile values which are in the same order we provided to the quantiles arguement in jags.predict.

```
wolfdeer.predict$expected.quantiles[1:10,]
```

```
##           [,1]     [,2]     [,3]
##  [1,] 28.03953 45.38741 74.76964
##  [2,] 26.71237 43.65384 71.52536
##  [3,] 26.28215 42.77669 69.41359
##  [4,] 26.29868 42.26658 68.28020
##  [5,] 25.20231 40.99472 67.27159
##  [6,] 24.45989 39.67519 65.32616
##  [7,] 24.02773 38.46369 63.36109
##  [8,] 23.40688 37.93704 61.12767
##  [9,] 22.57710 36.52470 59.98721
## [10,] 22.09413 35.75898 57.83802
```

JAGS Prediction Plotting Function

Now we can easily use this data to make a nice plot of the expected relationship between wolves and deer given our model. I've also made a handy function for this, **plot.jags.predict**!

```
plot.jags.predictions <-
  function(plotting.data,
           expected.covariate.values,
           quantiles,
           x.lab,
           y.lab,
           covariate.type = "continuous",
           category.names){

    if(covariate.type == "continuous"){

      smoothScatter(plotting.data$response ~ expected.covariate.values[plotting.data$cov],
                    las = 1, nrpoints = 0,
                    ylab = y.lab,
                    xlab = x.lab)
      lines(quantiles[,2] ~ expected.covariate.values, lty = 1, lwd = 3, col = 'white')
      lines(quantiles[,1] ~ expected.covariate.values, lty = 2, lwd = 3, col = 'white')
      lines(quantiles[,3] ~ expected.covariate.values, lty = 2, lwd = 3, col = 'white')

    } else if(covariate.type == "categorical"){

      plot.df <-
        data.frame(category = category.names,
                   cov = 1:nrow(quantiles),
                   low.CI = quantiles[,1],
                   median = quantiles[,2],
                   high.CI = quantiles[,3])

      ggplot() +
        geom_violin(data = plotting.data, aes(as.factor(cov), response), bw = 0.01, fill = "gray90") +
        geom_linerange(data = plot.df, aes(x = as.factor(cov), ymin = low.CI, ymax = high.CI), size = 0
        geom_point(data = plot.df, aes(as.factor(cov), median), color = "red", size = 2.5) +
        theme_bw() +
        scale_x_discrete(name = waiver(), labels = category.names) +
        ylab("Probability of Selection") +
        xlab("")

    } else {
      stop("covariate.type must be one of 'continuous' or 'categorical'")
    }

  }
```

Let's go over the arguments for this function:

plotting.data: The *plotting.data* data frame from jags.predict

expected.covariate.values: The expected covariate values which you will be plotting against. Only needed for covariate.type = "continuous". Make sure this is one of the items from the *expected.cov.values* list from jags.predict, and not the list itself.

quantiles: The *expected.quantiles* matrix from jags.predict. Make sure this is a matrix with length 3, ordered so column 1 is the lower credible interval, column 2 is the median (0.5), and column 3 is the upper credible interval.

x.lab: Character label for the x-axis

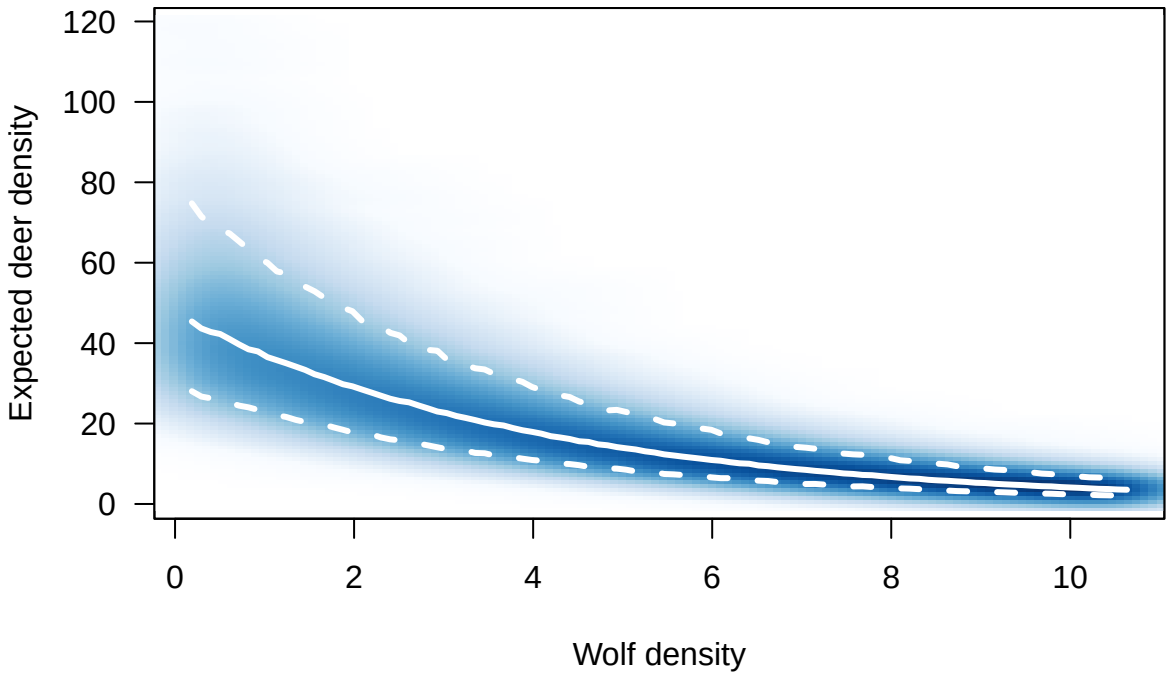
y.lab: Character label for the y-axis

covariate.type: Is the covariate being predicted "continuous" (default) or "categorical"

category.names The names of the continuous covariates in a character vector of the same order as your slopes list so labels can be correct. The first label should be for the intercept group. Only needed for covariate.type = "categorical".

The output of this is a nice simple plot:

```
plot.jags.predictions(plotting.data = wolfdeer.predict$plotting.data,
                      expected.covariate.values = wolfdeer.predict$expected.cov.values[[1]],
                      quantiles = wolfdeer.predict$expected.quantiles,
                      x.lab = "Wolf density",
                      y.lab = "Expected deer density")
```



The plot shows our simulated density in blue as the background, with a white solid line for the median and dashed lines for our upper and lower 95% credible intervals. Even if you dont want this exact plot, take a look at hat the inside of the function takes for plotting to understand how to plot based on jags.predict.

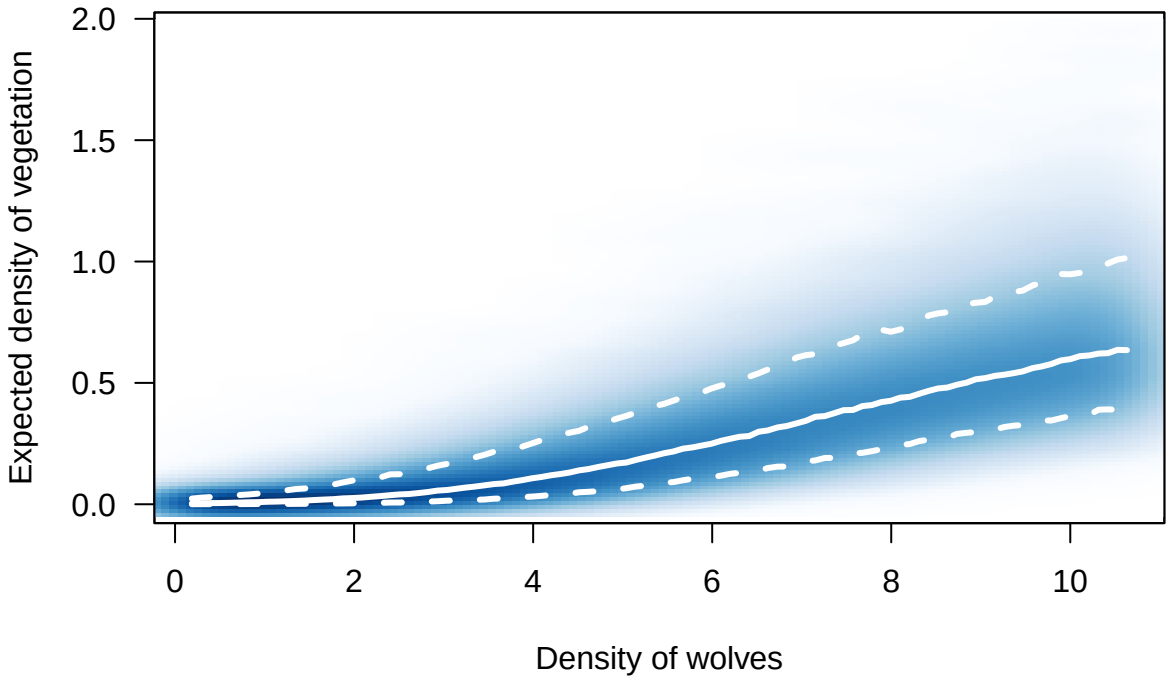
Now we'll see how to use predictions from one response of a model to inform predictions from a model in a path analysis framework.

We'll now predict vegetation as a function of our predicted deer values given our range of wolf densities. First let's extract what we need from the model, then run jags.predict

```
#same iterations as before
m2.intercepts <- model$sims.list$beta[,1]
m2.slopes <- list(model$sims.list$beta[,2])
m2.covs <- list(wolfdeer.predict$expected.response.matrix) #Notice here how instead of feeding a vector
m2.sd <- model$sims.list$sigma[,2]

deerveg.predict <-
jags.predict(intercepts = m2.intercepts,
             slope.coefficients = m2.slopes,
             exp.cov.values = m2.covs,
             res = 100,
             link.function = "lognormal",
             credible.intervals = c(0.05,0.5,0.95),
             lognormal.sd = m2.sd)
```

Now to plot for a path analysis my plot.jags.predict function doesn't work in its current form (future thing to be worked on), so instead we'll plot this manually to determine the effects of wolves on vegetation.

```
smoothScatter(deerveg.predict$plotting.data$response ~ wolfdeer.predict$expected.cov.values[[1]][deerveg
              las = 1, nrpoints = 0,
              ylab = 'Expected density of vegetation',
              xlab = 'Density of wolves')
lines(deerveg.predict$expected.quantiles[,2] ~ wolfdeer.predict$expected.cov.values[[1]], lty = 1, lwd =
lines(deerveg.predict$expected.quantiles[,1] ~ wolfdeer.predict$expected.cov.values[[1]], lty = 2, lwd =
lines(deerveg.predict$expected.quantiles[,3] ~ wolfdeer.predict$expected.cov.values[[1]], lty = 2, lwd =
```



Awesome, we can now see how vegetation density is predicted to change as wolf density increases.

Categorical Example

Below is example code of how this would work for a categorical model.

First let's look at the model structure. This is a model to predict deer resource selection as a function of land cover classes summarized from the National Land Cover Dataset and distance from road:

```
sink(file = "data/deer_rsf.jags")
cat("
    model {
    for(j in 1:7){
      beta[j] ~ dnorm(0, 0.1)
    }

    for(i in 1:n){
      logit(use[i]) <- beta[1] + beta[2] * crop[i] + beta[3] * developed[i] +
                       beta[4] * grassland[i] + beta[5] * water[i] + beta[6] * wetland[i] +
                       beta[7] * road.dist[i]

      y[i] ~ dbern(use[i])
    }

    }
", fill = TRUE)
sink()
```

For the sake of brevity, I'm not going to run the model here, but instead show an example of how the functions would work to make categorical predictions.

The key difference is in setting up the expected covariate values list, where every categorical expected covariate is labeled "category". Continuous covariates should be held at their mean. Category name labels start with the intercept group and otherwise stay in the same order as your slope list.

```
m.intercepts <- model$sims.list$beta[,1]

m.slopes <- list(model$sims.list$beta[,2],
                 model$sims.list$beta[,3],
                 model$sims.list$beta[,4],
                 model$sims.list$beta[,5],
                 model$sims.list$beta[,6],
                 model$sims.list$beta[,7])

m.covs <- list("category",
               "category",
               "category",
               "category",
               "category",
               mean(deer.final.df$road.dist))

m.category.names <- c("Forest", "Crop", "Developed", "Grassland", "Water", "Wetland")
```

You would then put everything into the functions below just like before and get a plot as output:

```
m.land.predict <-
  jags.predict(intercepts = m.intercepts,
               slope.coefficients = m.slopes,
               exp.cov.values = m.covs,
               link.function = "logit",
               covariate.type = "categorical")

plot.jags.predictions(plotting.data = m.land.predict$plotting.data,
                      quantiles = m.land.predict$expected.quantiles,
                      x.lab = "",
                      y.lab = "Probability of Selection",
                      covariate.type = "categorical",
                      category.names = m.categroy.names)
```